Compilers

Dr. Sherin ElGokhy

Lecture#1

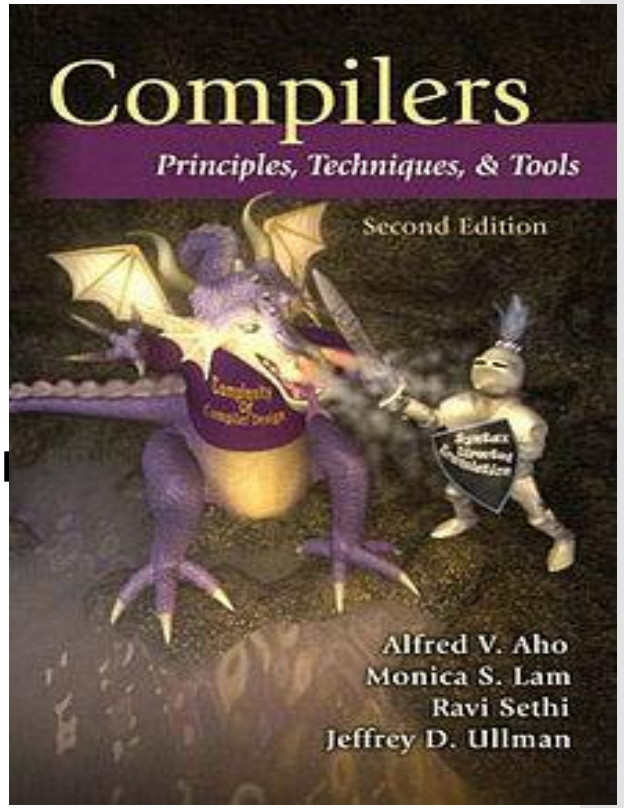
Reference

- The Purple Dragon Book

Compilers: Principles, Techniques, and Tools

- Aho, Lam, Sethi & Ullman

- Not required
  - But a useful reference



Course Structure

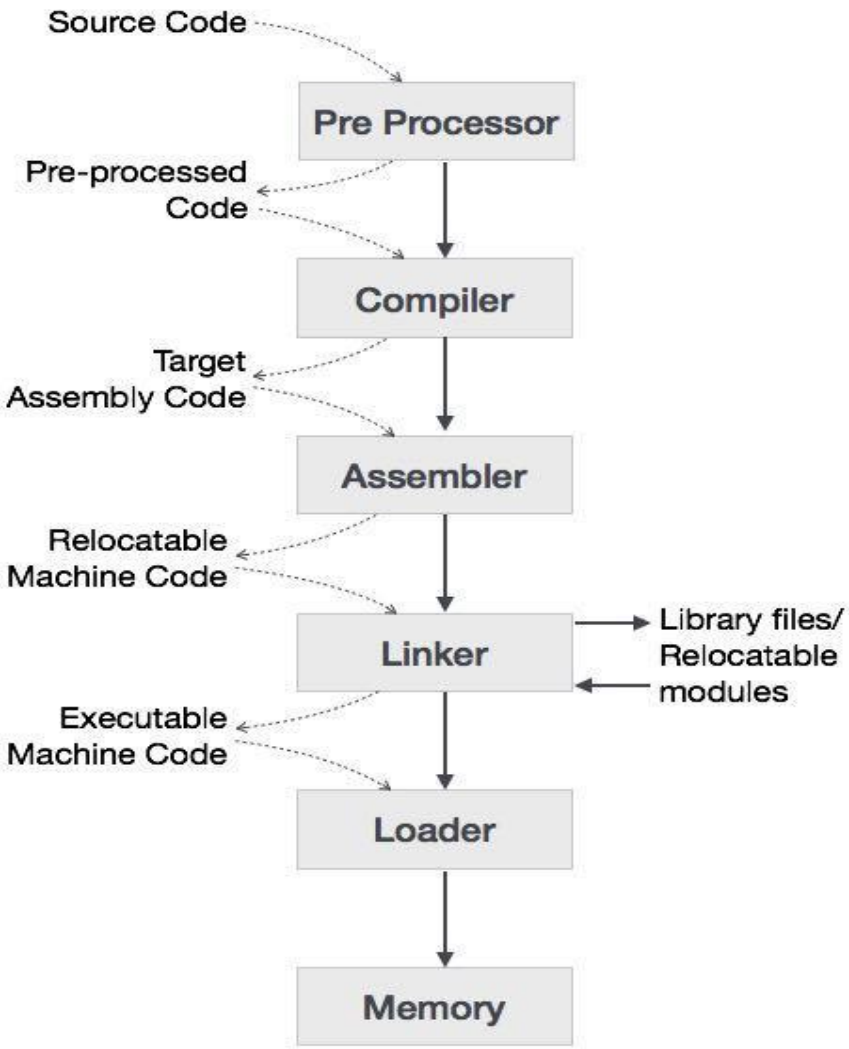
- **Course has theoretical and practical aspects**
- **Written assignments = theory**
- **Programming project**

A big project… in 4 easy parts
Start early!

Language Processing System

- Any computer system is made of hardware and software.
- The hardware understands a language, which humans cannot understand. Hardware understands instructions in the form of electronic charge, which is the counterpart of binary language. Binary language has only two alphabets, 0 and 1.
- So we write programs in high-level language, which is easier for us to understand and remember.
- These programs are then fed into a series of tools and OS components to get the desired code that can be used by the machine.
- This is known as **Language Processing System**.

Language Processing System



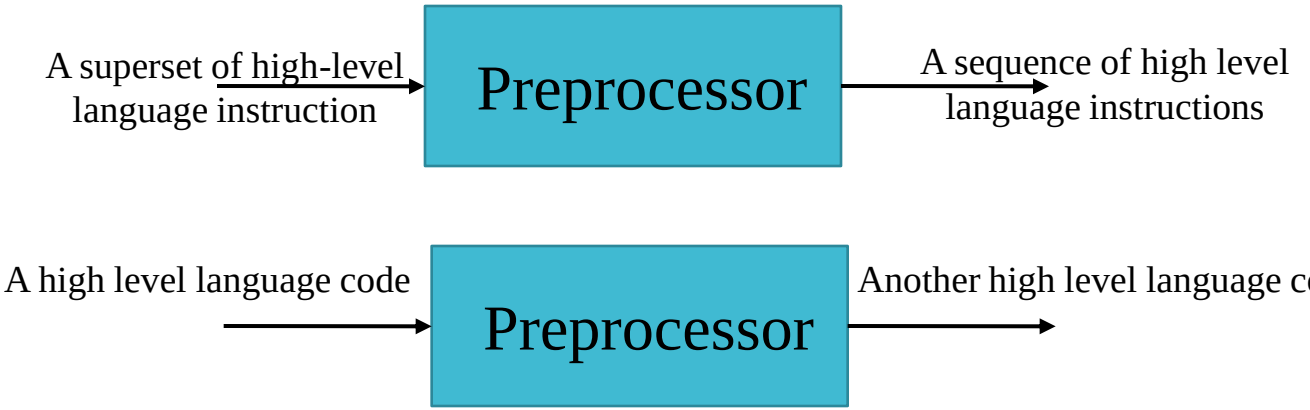
Translators

- Translators are programs which accept a textual representation of an algorithm expressed in a source language, and produce as a primary output a representation of the same algorithm expressed in another language (The target language).

Types of Translators

- **Preprocessor**

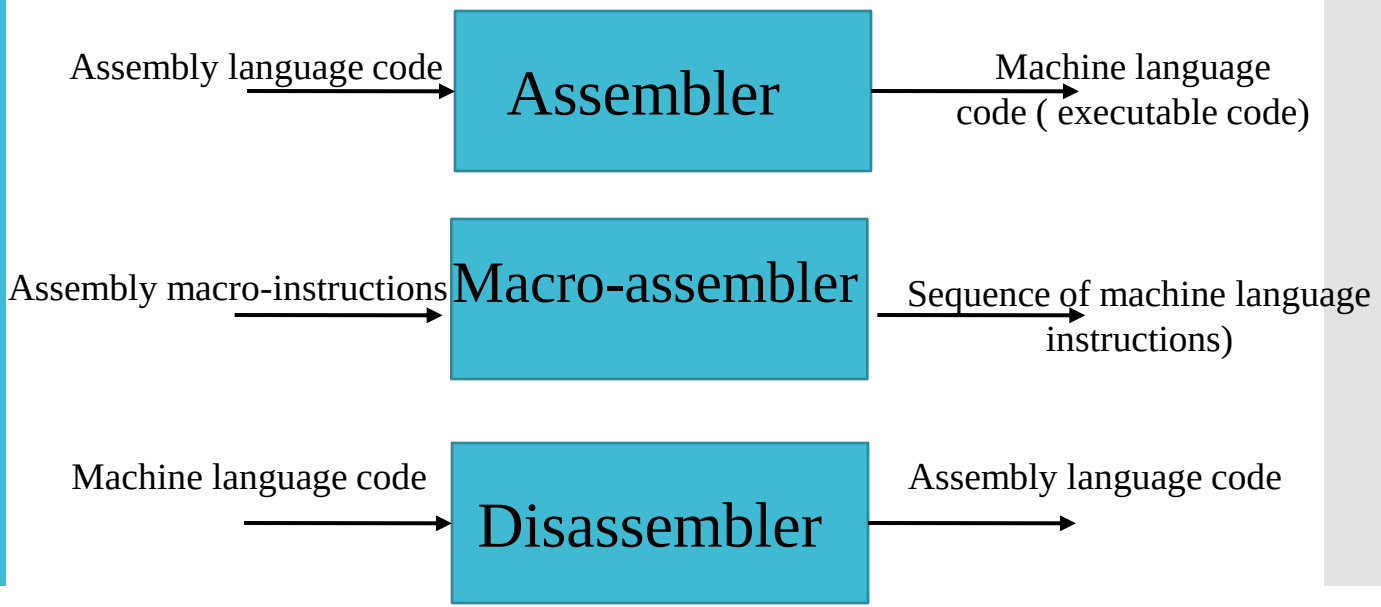
A preprocessor, generally considered as a part of compiler, is a tool that produces input for compilers. It deals with macro-processing, file inclusion, language extension, etc.



Types of Translators

- **Assembler**

An assembler translates assembly language programs into machine code. The output of an assembler is called an object file, which contains a combination of machine instructions as well as the data required to place these instructions in memory.



Types of Translators

- **Interpreter**

An interpreter, like a compiler, translates high-level language into low-level machine language.

The difference between compiler and interpreter

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python use interpreters.	Programming language like C, C++ use compilers.

Compilers

- **Compiler**

A compiler is a program that can read a program in one language — the *source language — and translate it into an equivalent program in* another language — the *target language.*

- *An important role of the* compiler is to report any errors in the source program that it detects during the translation process.

Compilers

- **Cross-compiler**

A compiler that runs on platform (A) and is capable of generating executable code for platform (B) is called a cross-compiler.

- **Source-to-source Compiler**

A compiler that takes the source code of one programming language and translates it into the source code of another programming language is called a source-to-source compiler.

- **Decompiler**

A program that translates from a low level language (machine language) to a higher level one (high level language).

Classification of compilers

According to…….

construction: how they have been constructed?

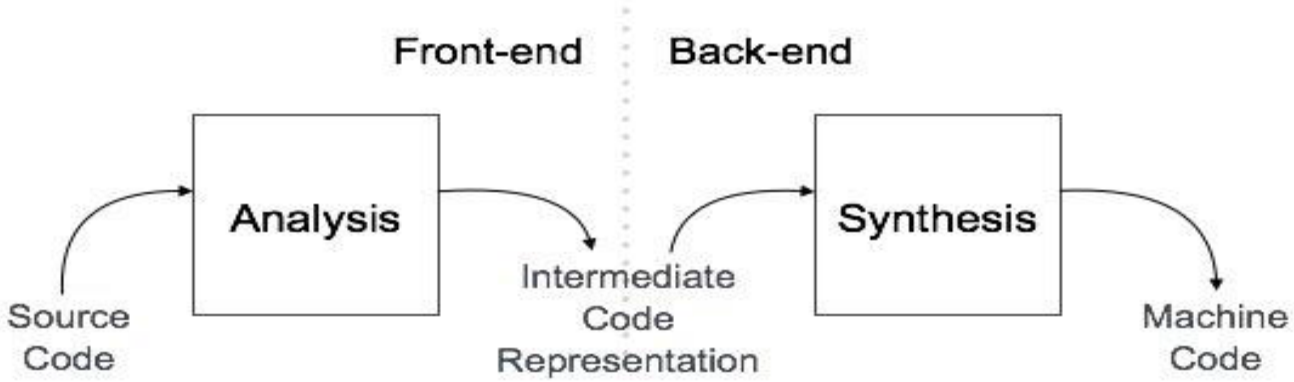
- Single pass
- Multiple Pass
- Load &Go

function: what functions they are supported to perform?

- Debugging
- Optimization

Compiler Architecture

A compiler can broadly be divided into two phases based on the way they compile.



Compiler Architecture

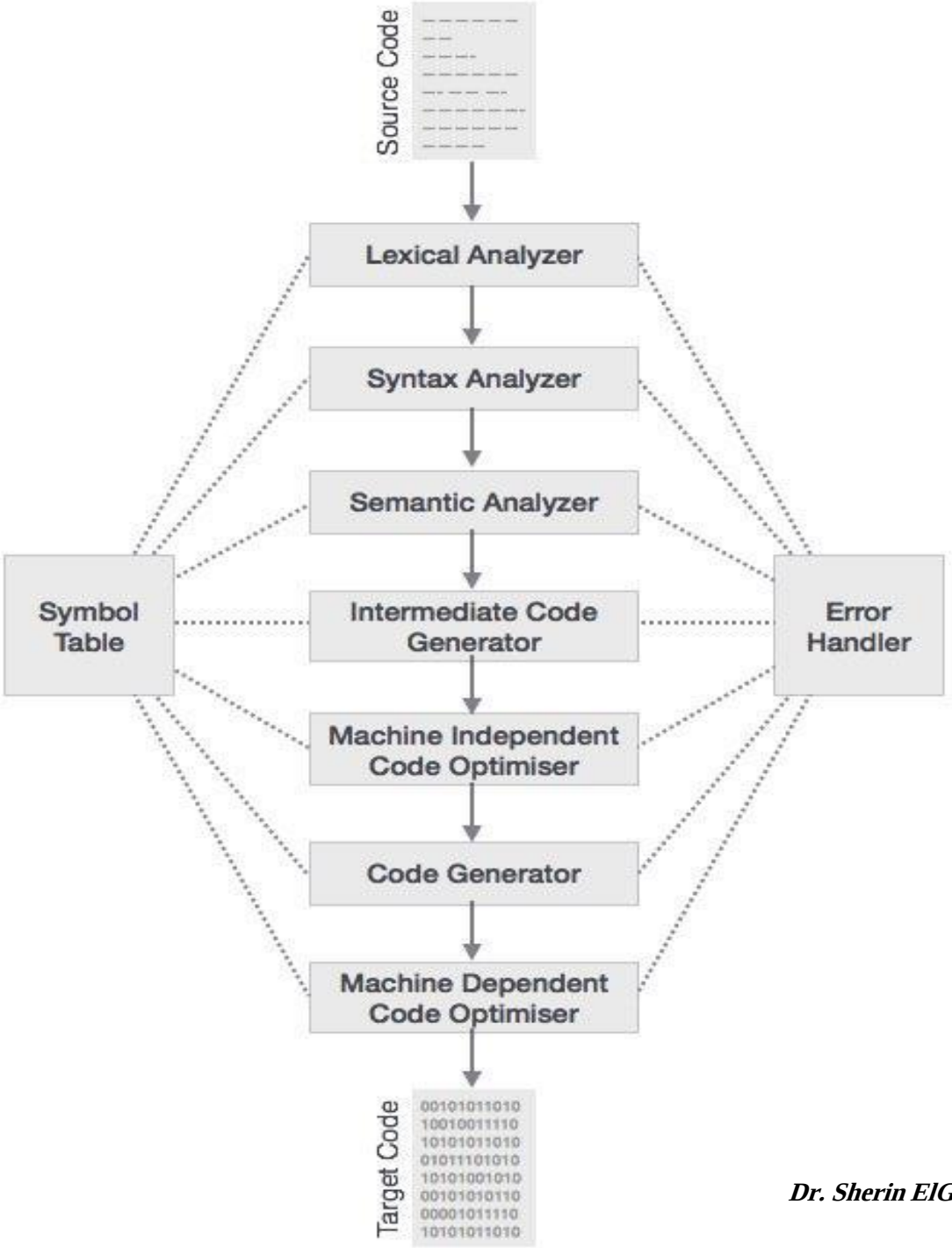
- **Analysis Phase**

Known as the front-end of the compiler, the analysis phase of the compiler reads the source program, divides it into core parts, and then checks for lexical, grammar, and syntax errors. The analysis phase generates an intermediate representation of the source program and symbol table, which should be fed to the Synthesis phase as input.

- **Synthesis Phase**

Known as the back-end of the compiler, the synthesis phase generates the target program with the help of intermediate source code representation and symbol table.

The Structure of a Compiler



Lexical Analysis

- **Lexical Analysis**

The first phase of a compiler –lexical analyzer or scanner- works as a text scanner. This phase scans the source code as a stream of characters and converts it into meaningful lexemes. Lexical analyzer represents these lexemes in the form of tokens as:

<token-name, attribute-value>

- First step: recognize words.
  - Smallest unit above letters

This is a sentence.

Lexical Analysis

- Lexical analysis is not trivial. Consider:

ist his ase nte nce

Lexical Analysis

Lexical analyzer divides program text into "words" or "tokens"

If x == y then z = 1; else z = 2;

- Units:

Syntax Analysis or Parsing

- **Syntax Analysis**

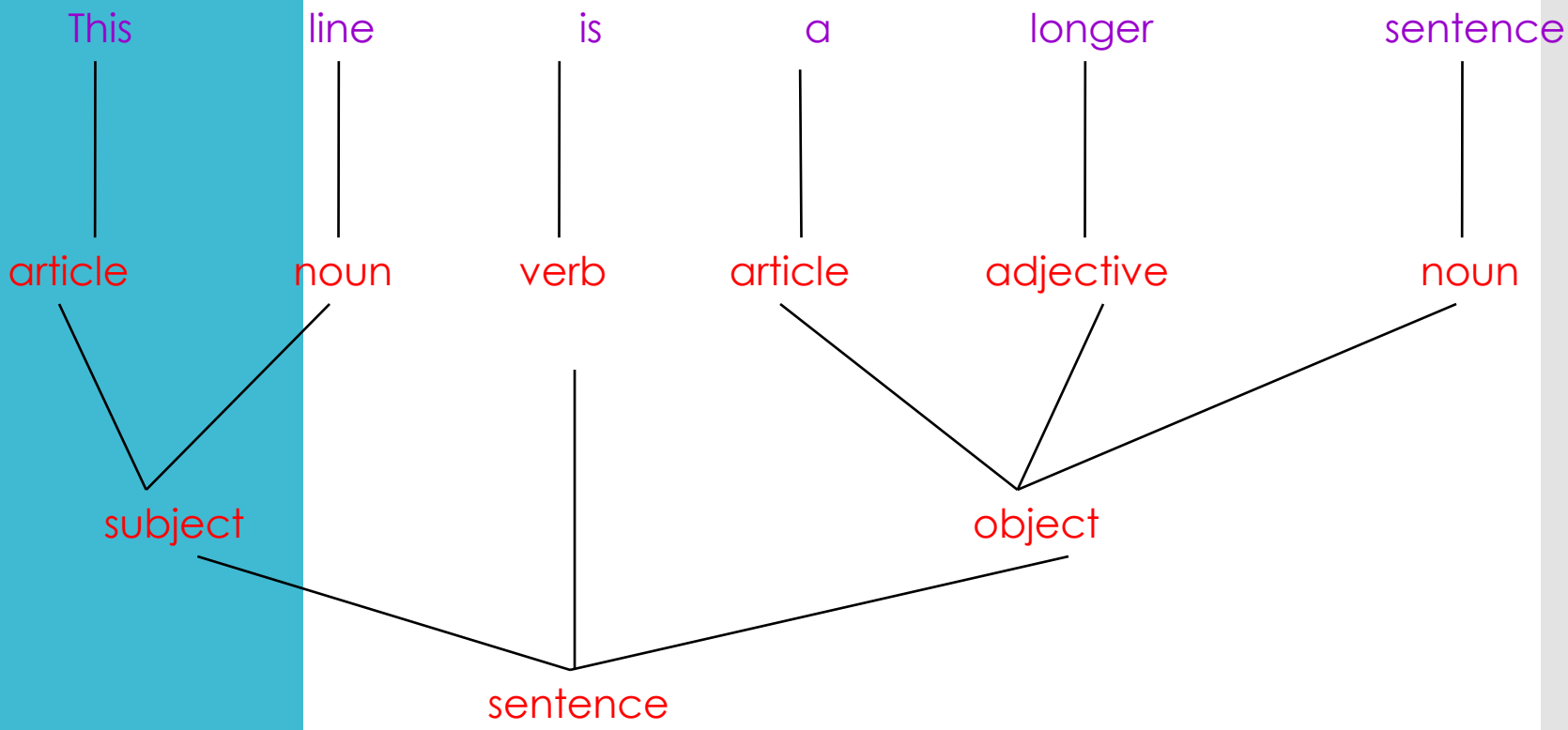
The next phase is called the syntax analysis or parsing. It takes the token produced by lexical analysis as input and generates a parse tree (or syntax tree).

In this phase, token arrangements are checked against the source code grammar, i.e., the parser checks if the expression made by the tokens is syntactically correct

- Once words are understood, the next step is to understand sentence structure

- Parsing = Diagramming Sentences
  - The diagram is a tree

Diagramming a Sentence

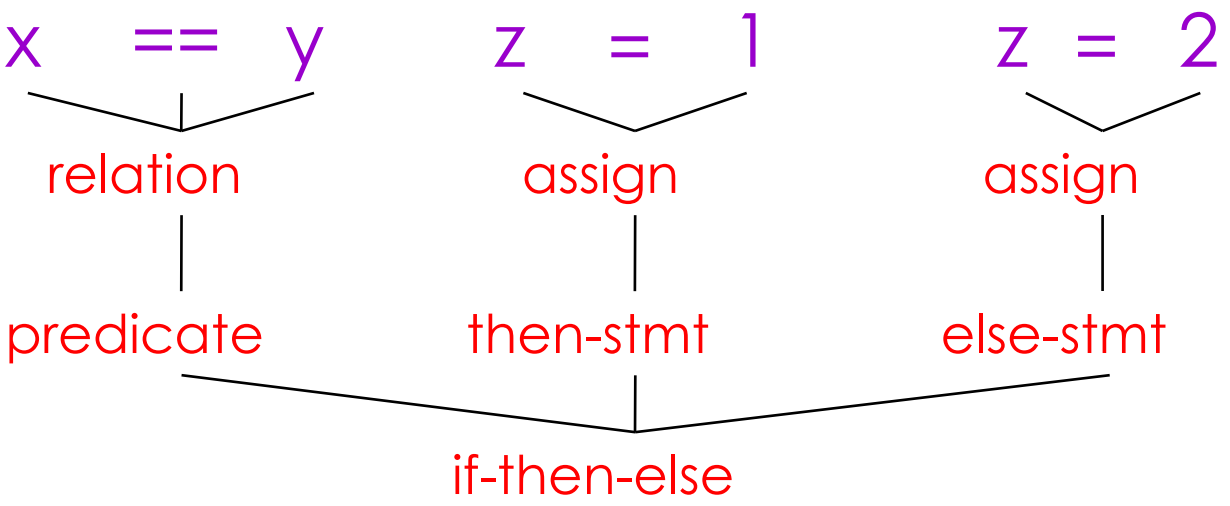


Parsing Programs

- Parsing program expressions is the same
- Consider:

If x == y then z = 1; else z = 2;

- Diagrammed:



Semantic Analysis

- Once sentence structure is understood, we can try to understand "meaning"
  - But meaning is too hard for compilers

- Compilers perform limited analysis to **catch inconsistencies**
- Semantic analysis checks whether the parse tree constructed follows the rules of language.

For example: assignment of values is between compatible data types, and adding string to an integer. Also, the semantic analyzer keeps track of identifiers, their types and expressions; whether identifiers are declared before use or not, etc. The semantic analyzer produces an annotated syntax tree as an output.

Semantic Analysis in English

- Example:

  Jack said Jerry left his assignment at home.

  What does “his” refer to? Jack or Jerry?

- Even worse:

  Jack said Jack left his assignment at home?

  How many Jacks are there?

  Which one left the assignment?

Semantic Analysis in Programming

- Programming languages define strict rules to avoid such ambiguities
- This C++ code prints "4"; the inner definition is used

```
{
  int Jack = 3;
  {
        int Jack = 4;
        cout <<
  Jack;
  }
}
```

More Semantic Analysis

Compilers perform many semantic checks besides variable bindings

Example:

Jack left her homework at home.

A "type mismatch" between her and Jack; we know they are different people

- Presumably Jack is male

Optimization

- No strong counterpart in English, but a little bit like editing
- Optimization can be assumed as something that removes unnecessary code lines, and arranges the sequence of statements in order to speed up the program execution without wasting resources (CPU, memory).

- Automatically modify programs so that they
  - Run faster
  - Use less memory
  - In general, conserve some resource

Optimization Example

X = Y * 0 is the same as X = 0

Code Generation

- Produces assembly code (usually)
- In this phase, the code generator takes the optimized representation of the intermediate code and maps it to the target machine language. The code generator translates the intermediate code into a sequence of (generally) re-locatable machine code. Sequence of instructions of machine code performs the task as the intermediate code would do.

- A translation into another language
  - Analogous to human translation

Intermediate Languages

- After semantic analysis, the compiler generates an intermediate code of the source code for the target machine.
- It represents a program for some abstract machine.
- It is in between the high-level language and the machine language.
- This intermediate code should be generated in such a way that it makes it easier to be translated into the target machine code.

Symbol Table

- It is a data-structure maintained throughout all the phases of a compiler. All the identifiers' names along with their types are stored here. The symbol table makes it easier for the compiler to quickly search the identifier record and retrieve it. The symbol table is also used for scope management.

Compilers Today

- The overall structure of almost every compiler adheres to our outline
- The proportions have changed since FORTRAN
  - Early: lexing, parsing most complex, expensive
  - Today: optimization dominates all other phases, lexing and parsing are cheap

Thanks